Course FSD311: Generative Adversarial Networks (GANs)

Thomas PIERROT
t.pierrot@instadeep.com

Toulouse, December 4th 2018

Contents

Motivation

Yann Lecun (Director of Facebook AI Research Paris and Professor at NYU) on Twitter:

> "There are many interesting recent development in deep learning, probably too many for me to describe them all here. But there are a few ideas that caught my attention enough for me to get personally involved in research projects. **The most important one, in my opinion, is adversarial training** (also called GAN for Generative Adversarial Networks). This is an idea that was originally proposed by Ian Goodfellow when he was a student with Yoshua Bengio at the University of Montreal (he since moved to Google Brain and recently to OpenAI). **This, and the variations that are now being proposed is the most interesting idea in the last years in ML, in my opinion**."

Motivation

Are those pictures real ?

Motivation

Are those pictures real ?

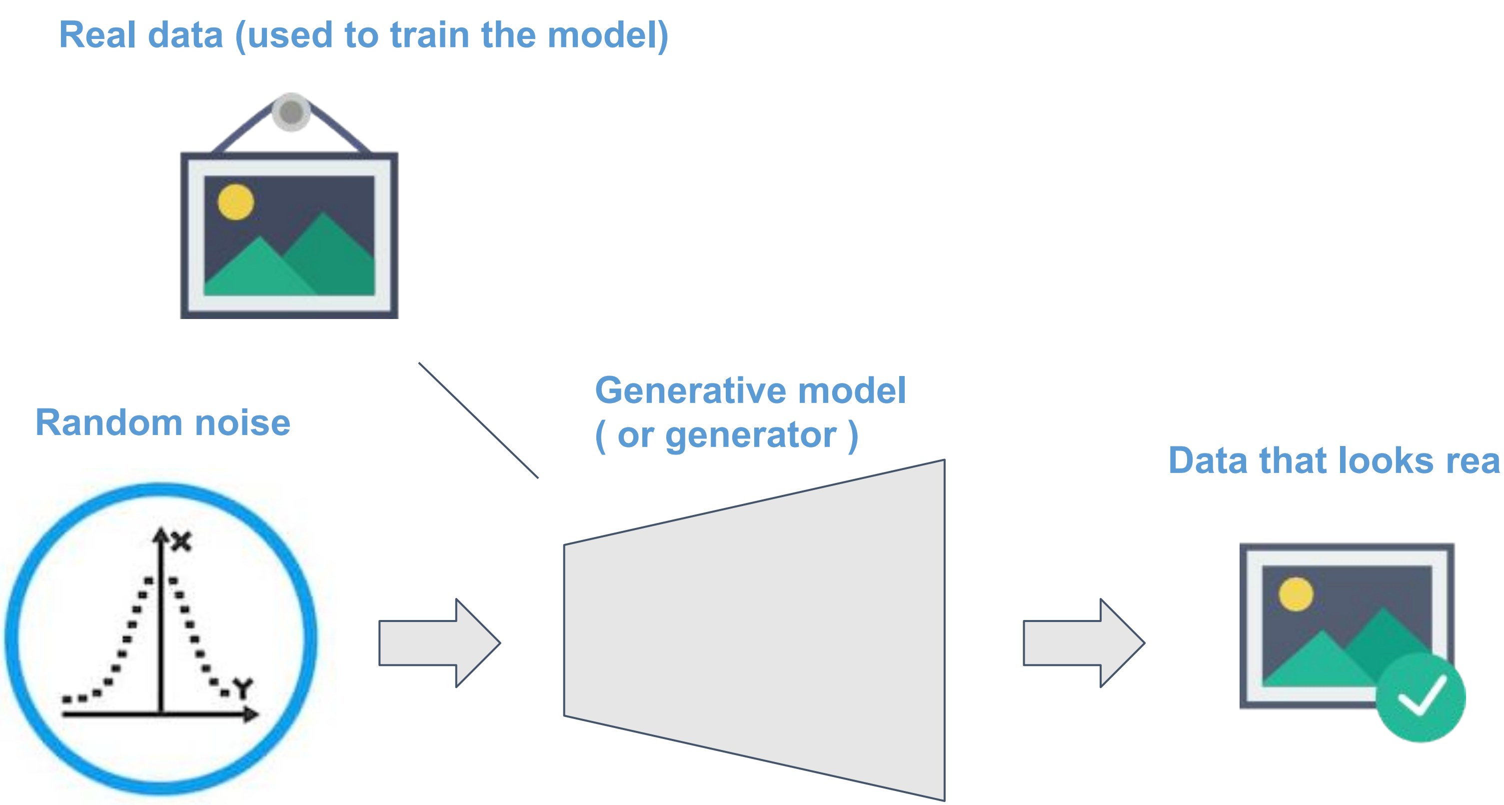
No ! They’ve been dreamed by a neural network.

Source : Progressive growing of gans paper

Generative Models

- Goal : Generate with a model (in this case a NN) data that look real but does not already exist.
- Example : Generate pictures that look like pictures from a given dataset.
- Problem : No natural input in this situation.
- Solution: Random numbers as inputs.
- Finally : Model that maps a given distribution (ex: gaussian) towards our data distribution.

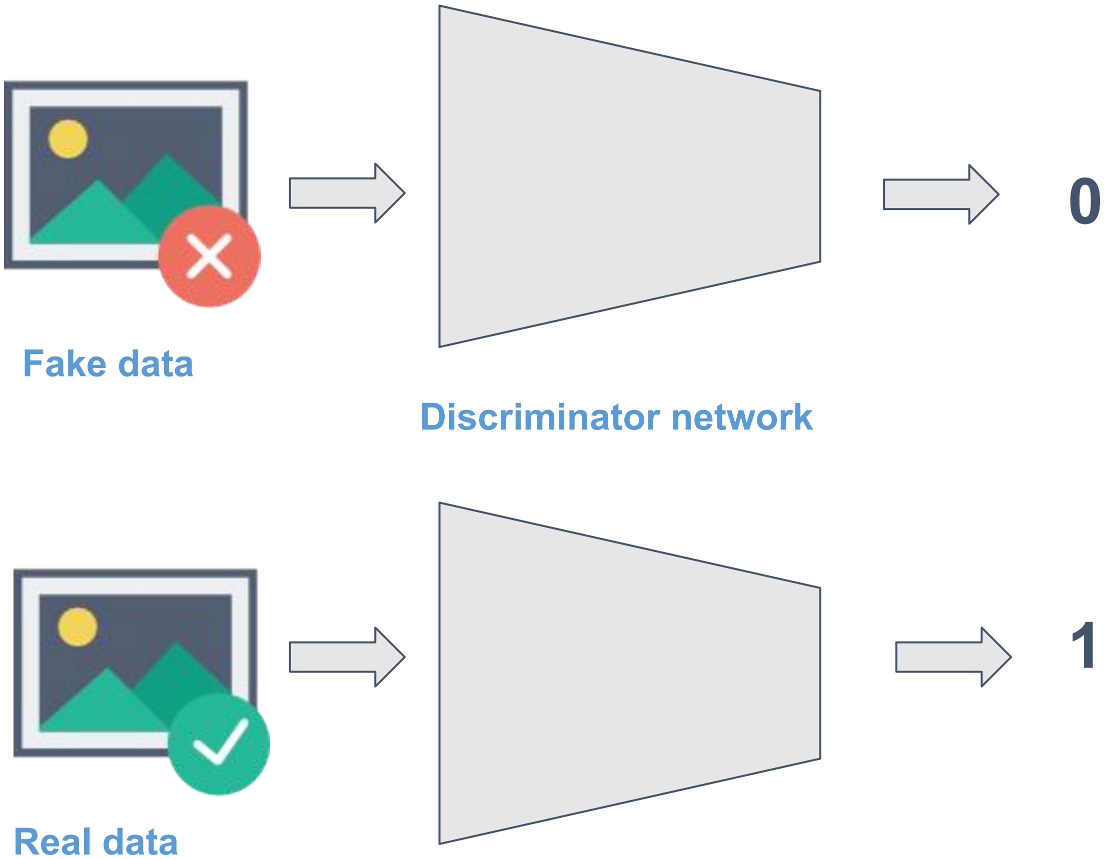
Generative Models



Generative Models

- **Question : How to define a meaningful loss to train the model ?**
- **Possible solution : Minimize the distance between the generator output and its closest neighbour in the dataset.**
- **Issue : Extremely expensive and leads to poor results.**
- **Good solution : Use a second network to train the generator.**

Discriminator network principle

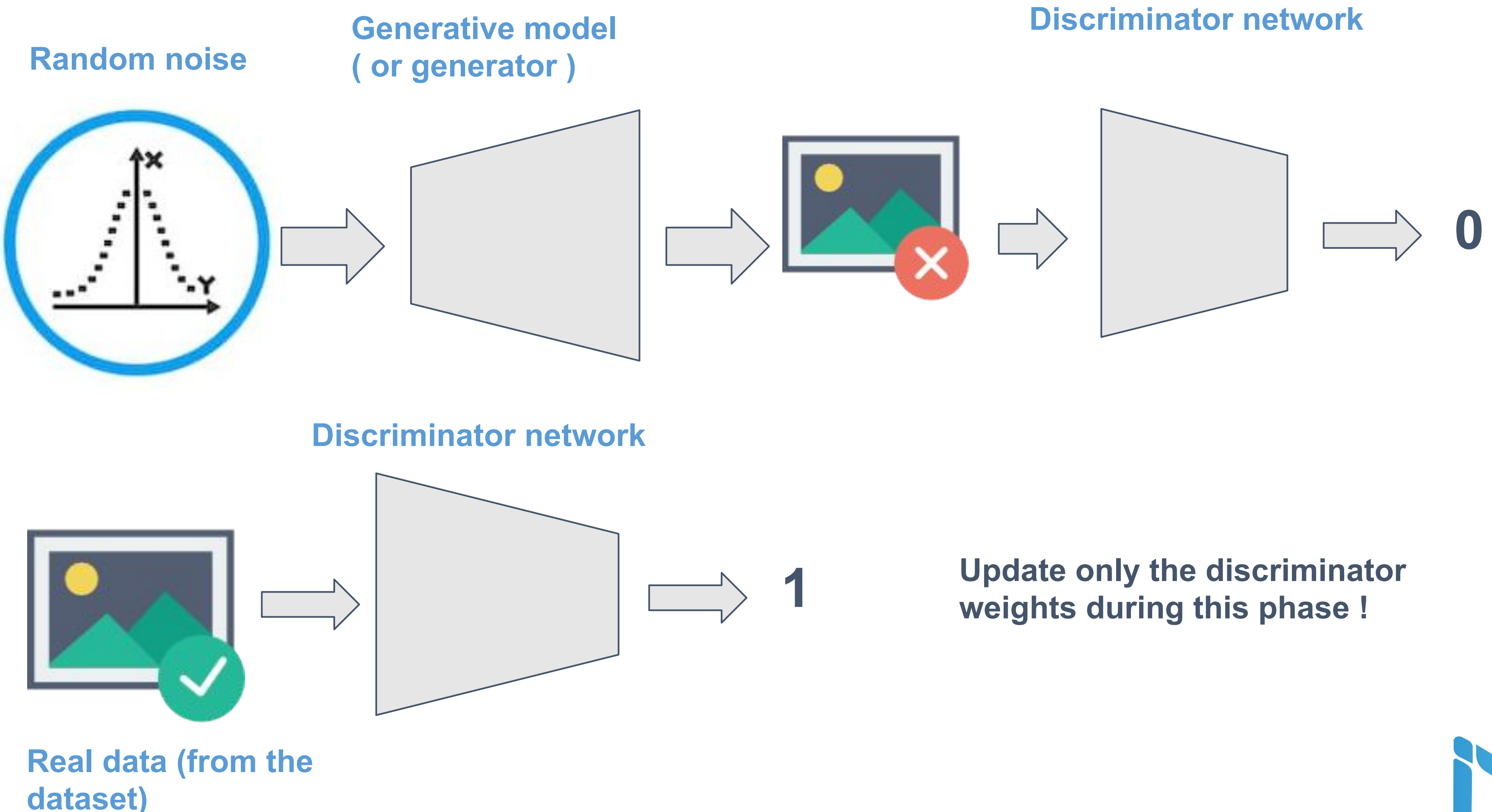


The discriminator returns the probability for a given input to be real.

Adversarial training

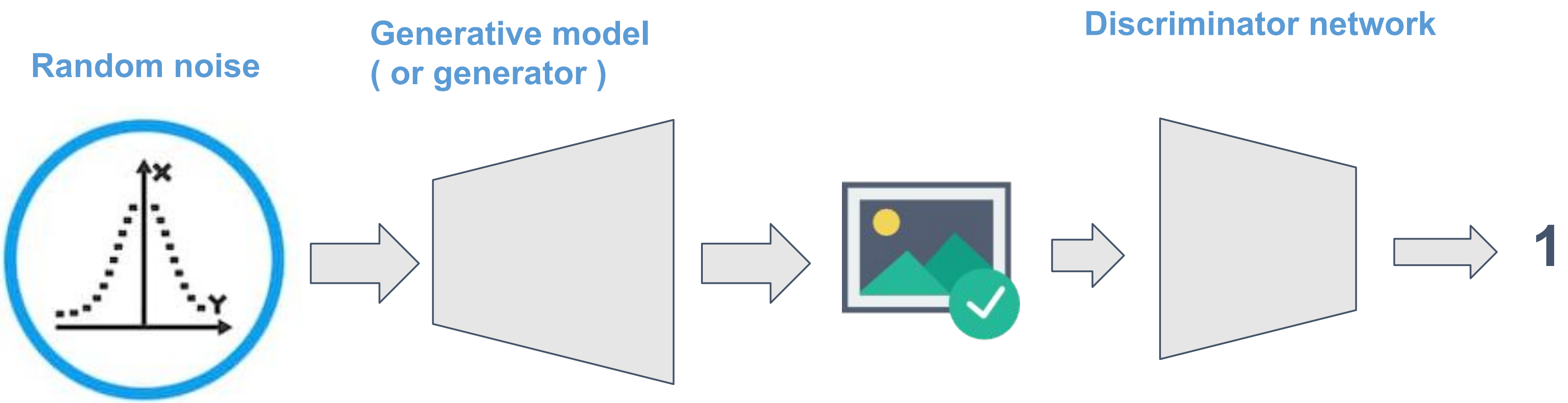
- Train the discriminator to make the difference between fake and real data (standard supervised training).
- Train the generator to fool the discriminator.
- Repeat it until the discriminator is completely lost (returns always 0.5).
- At this point the generator returns data that looks real.

Train the discriminator



Update only the discriminator
weights during this phase !

Train the generator



Update only the generator weights during this phase !

Let's write it mathematically !

Notations

- **Inputs :** $\mathbf{x} \in \mathcal{X}$
- **Random noise :** $\mathbf{z} \in \mathcal{G}, \quad \text{example} : \mathbf{z} \sim \mathcal{N}(0,1)$
- **Generator :** $G \;:\; \mathcal{G} \longrightarrow \mathcal{X}$
- **Discriminator :** $D \;:\; \mathcal{X} \longrightarrow \mathbb{R}$

How it works

- **Discriminator loss:**

$$L_D = -\sum_{i=1}^{n} \underbrace{\log\left(1 - D(G(z_i))\right)}_{L_D^{false}} + \underbrace{\log\left(D(x_i)\right)}_{L_D^{true}}$$

- **Generator loss:**

$$L_G = \sum_{i}^{n} \log\left(1 - D(G(z_i))\right)$$

Warning: I took the convention where the losses are to be minimized.

Note: As usual, the full sum is replaced with a sub-sum: we use mini-batches to compute the gradients.

Practical algorithm

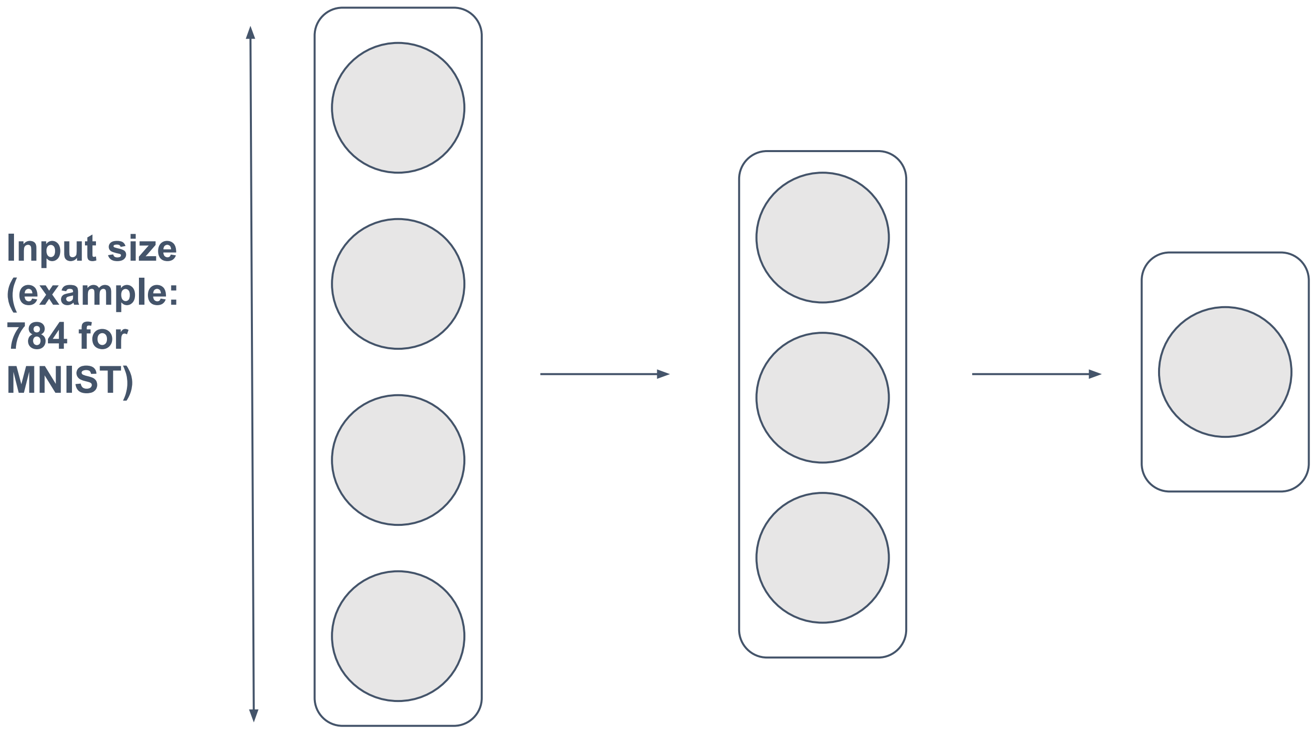
- **Initialize both networks**

- **Training:**
  - Sample a mini-batch of noise vectors and sample a mini-batch of input data.
  - Update the discriminator weights with those batches (gradient descent on its loss).
  - Sample a mini-batch of noise vectors.
  - Update the generator weights with this batch (gradient descent as well).

- **Repeat training phase until convergence**

Architectures used in the original paper

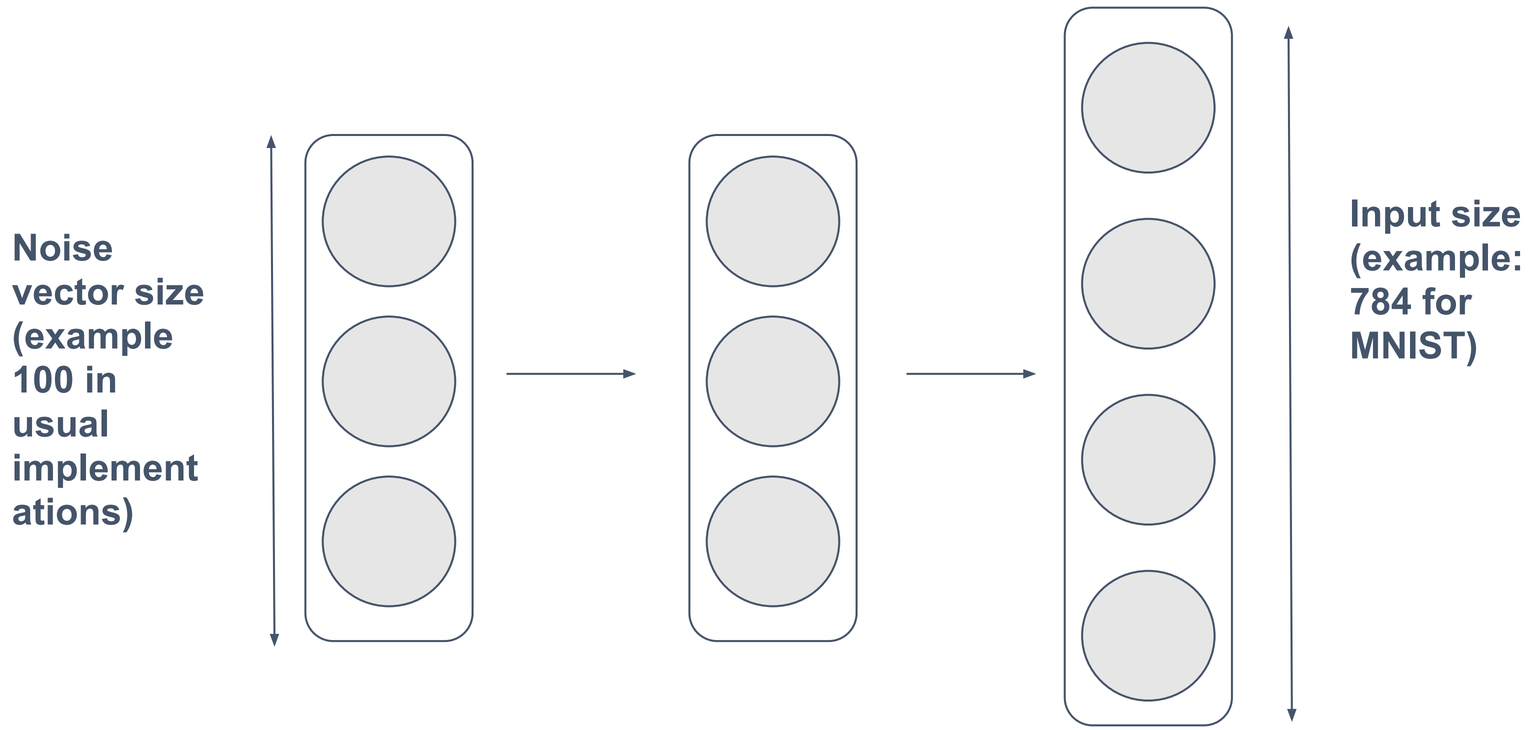
The discriminator (dense NN)



Takes a flattened image and returns a scalar value

Architectures used in the original paper

The generator (dense NN)



Takes a vector of random number and returns a flattened image

Toward the DCGANs

- **Now we want to work with large images.**

- **Problem: feed directly a large image into a dense NN will make the number of weights blow.**

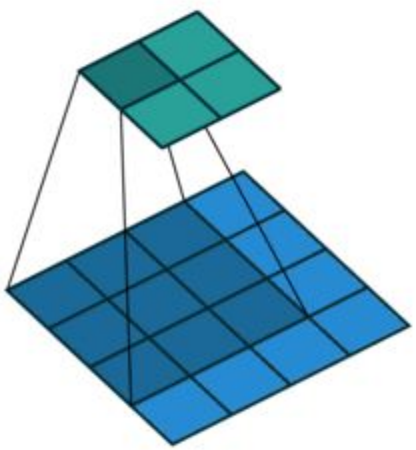
- **Standard idea: use a convolutional NN instead of a fully dense NN.**

Reminder : convNets

- Main idea : reduce input size (extract its features) before feeding it into a standard (dense) NN.

- Historically : hand-made feature engineering.

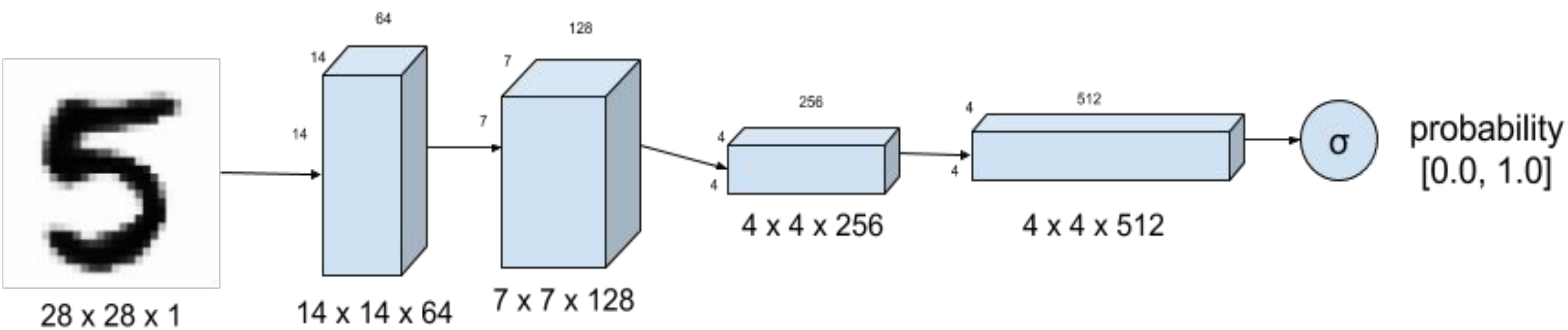
- Since 2012 : automatic feature extraction with convolution layers.



Example of convolution with a filter of size 3x3 on a 4x4 image. Result of the a convolution: a 2x2 image.

Example for the discriminator

- Extract the input image features through several convolutional layers.
- Use the extracted features inside a NN layer to determine whether the image is real or fake.



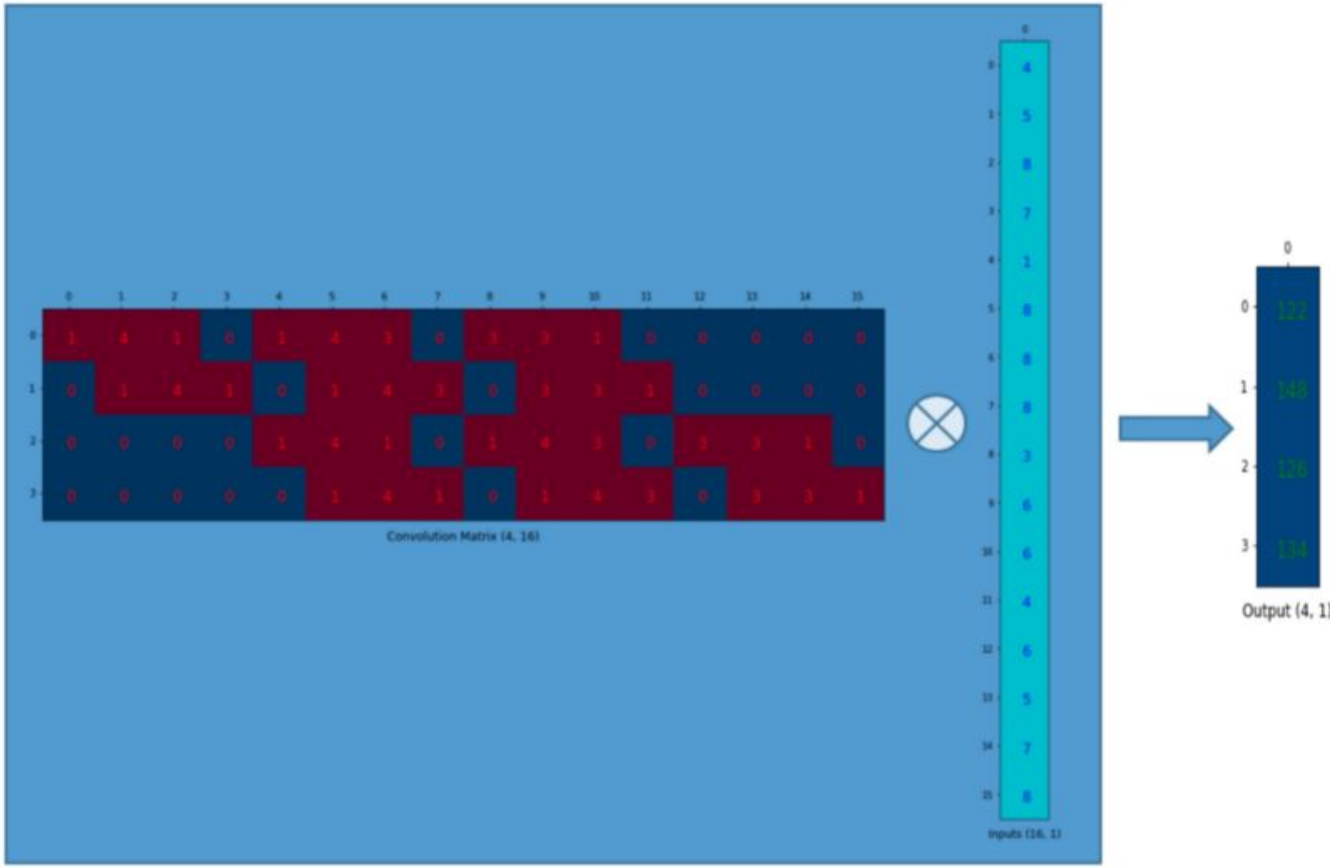
A discriminator example on MNIST

What about the generator ?

- Convolution decreases the data size: it is perfect for the discriminator.
- However in the generator we need an operation that increases the data size !
- One might use standard interpolation techniques.
- But it leads to poor results (they are not really “learnable”).
- Answer: the transposed convolution (also called deconvolution or up-sampling).

Toward the transposed-convolution

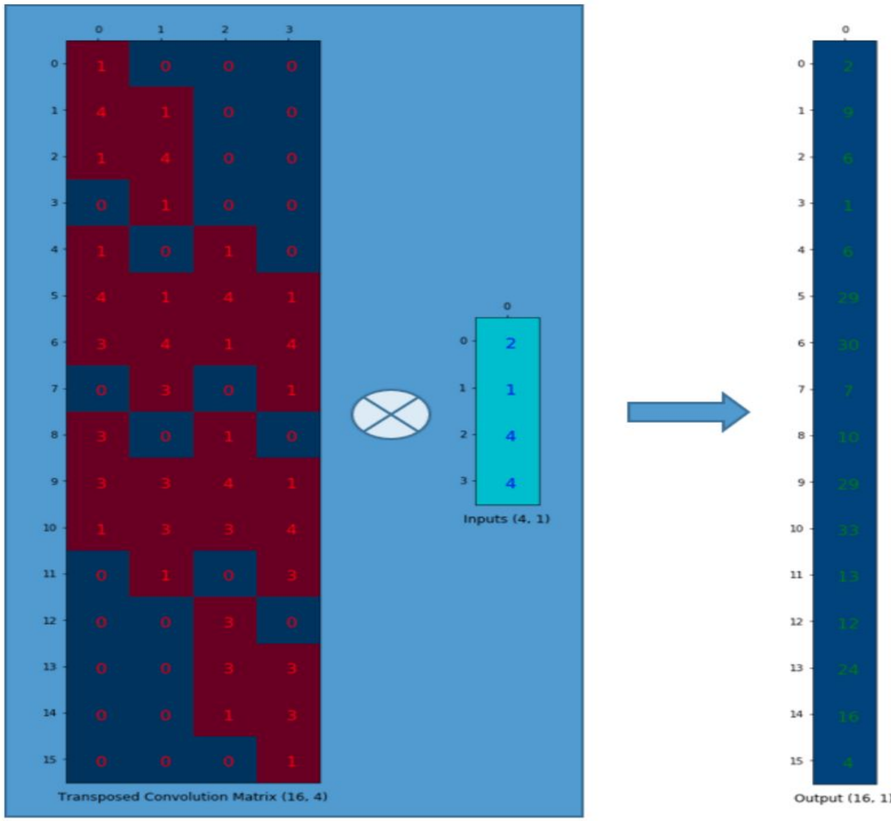
Convolution as a matrix vector product operation.



Example: convolution over a 4x4 image (flattened as a vector of size 16) with a 3x3 filter. It gives a 2x2 image

Toward the transposed-convolution

We define the inverse operation: matrix vector product with the transposed convolution matrix.

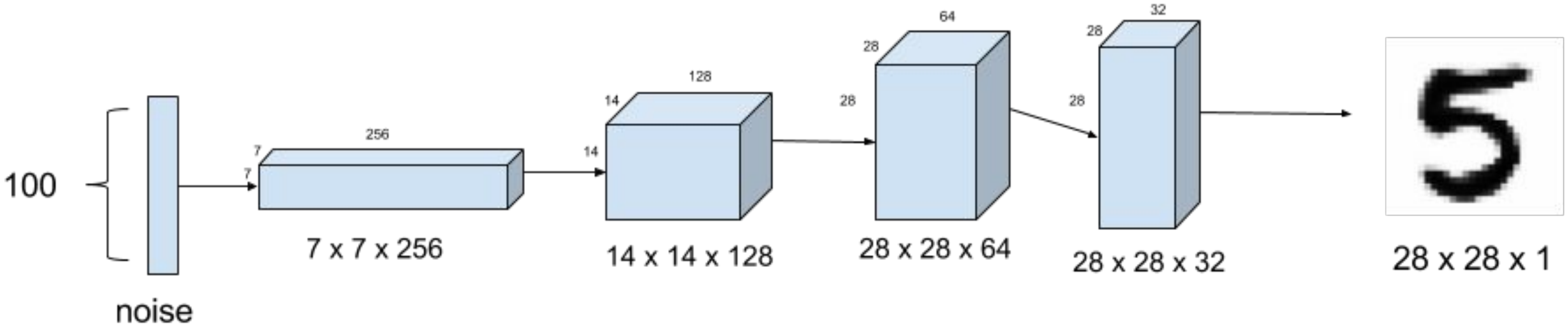


As for the convolution, the matrix coefficients are learnt!

Example: deconvolution over a 2x2 image (flattened as a vector of size 4) with a 3x3 filter. It gives a 4x4 image

Example for the generator

- **Perform up-sampling operations to transform the input noise vector into an image.**



A generator example on MNIST

DCGAN

- **Radford *et al.* (2015) present the Deep Convolutional GAN (DCGAN).**

Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

- **They generate fake pictures of bedrooms to demonstrate their architecture efficiency.**

SRGAN

- Ledig *et al.* (2016): **super-resolution GAN model, SRGAN that can take a low resolution photo and generate a high-resolution sample.**

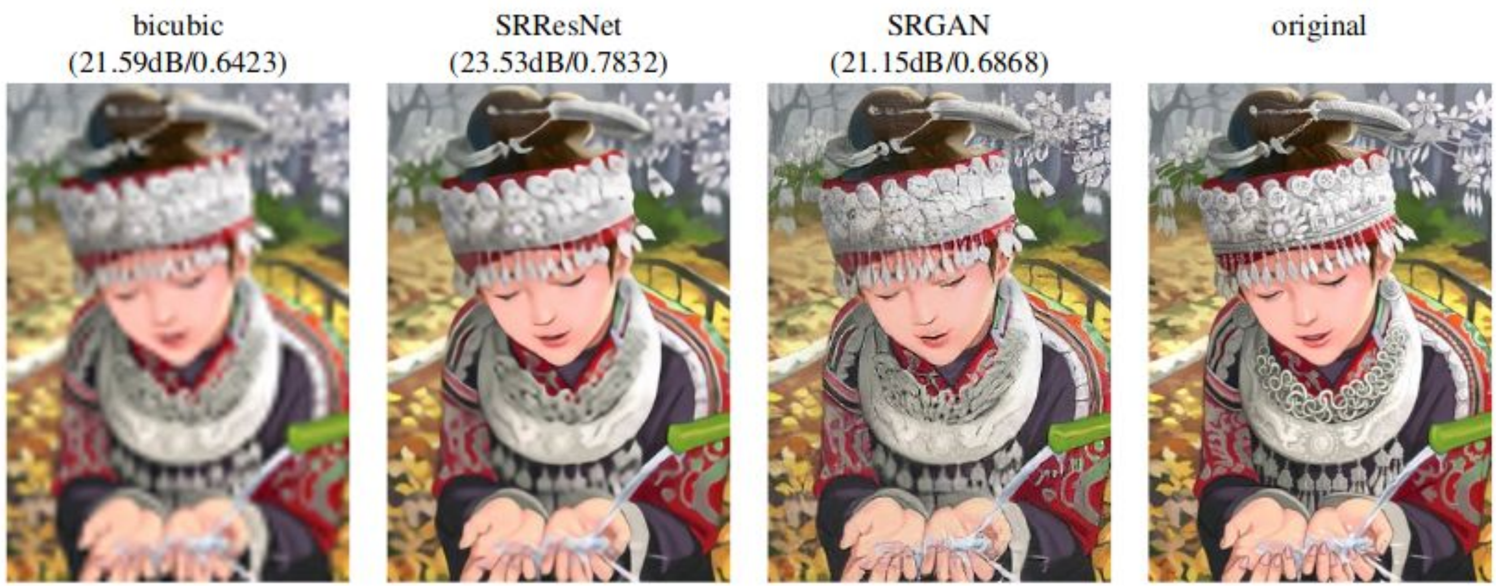


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

- **This problem is classical supervised learning. Here, the GAN scheme is used as an alternative loss to the standard RMS (or L2) loss.**

Conditional generative models

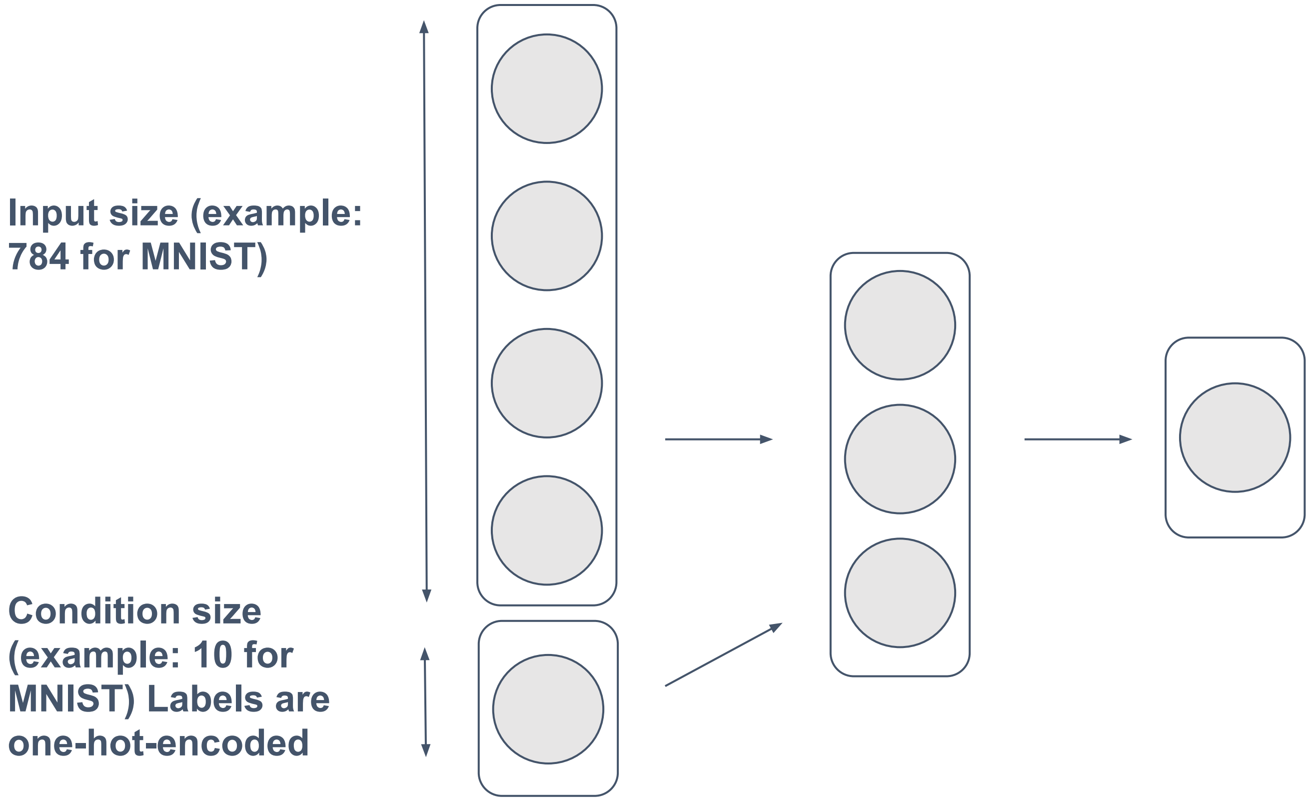
- GANs drawback: it generates data without “control on the output”.
- Example: if it learns to generate MNIST digits, it will generate any digit without distinction.
- One might desire to generate (with the same network) a given digit (a given label).
- Answer: we can condition the network output.
- In this case, we want to generate data given a condition c.
- Example: generate a MNIST digit given its label.

Conditional GANs

- **Adapt GAN to add a condition on the input: CGANs** (M. Mirza et al., 2014)
- **Simple idea: add an input to both the generator and the discriminator.**
- **The generator takes a noise vector and a condition (example a label) and generates conditioned (by this label) image.**
- **The discriminator takes an image and the condition and returns the conditioned (by this label) probability that this image is real.**
- **Note: most of the time the condition is a label, but it can be a description or other type of data.**

CGAN architecture

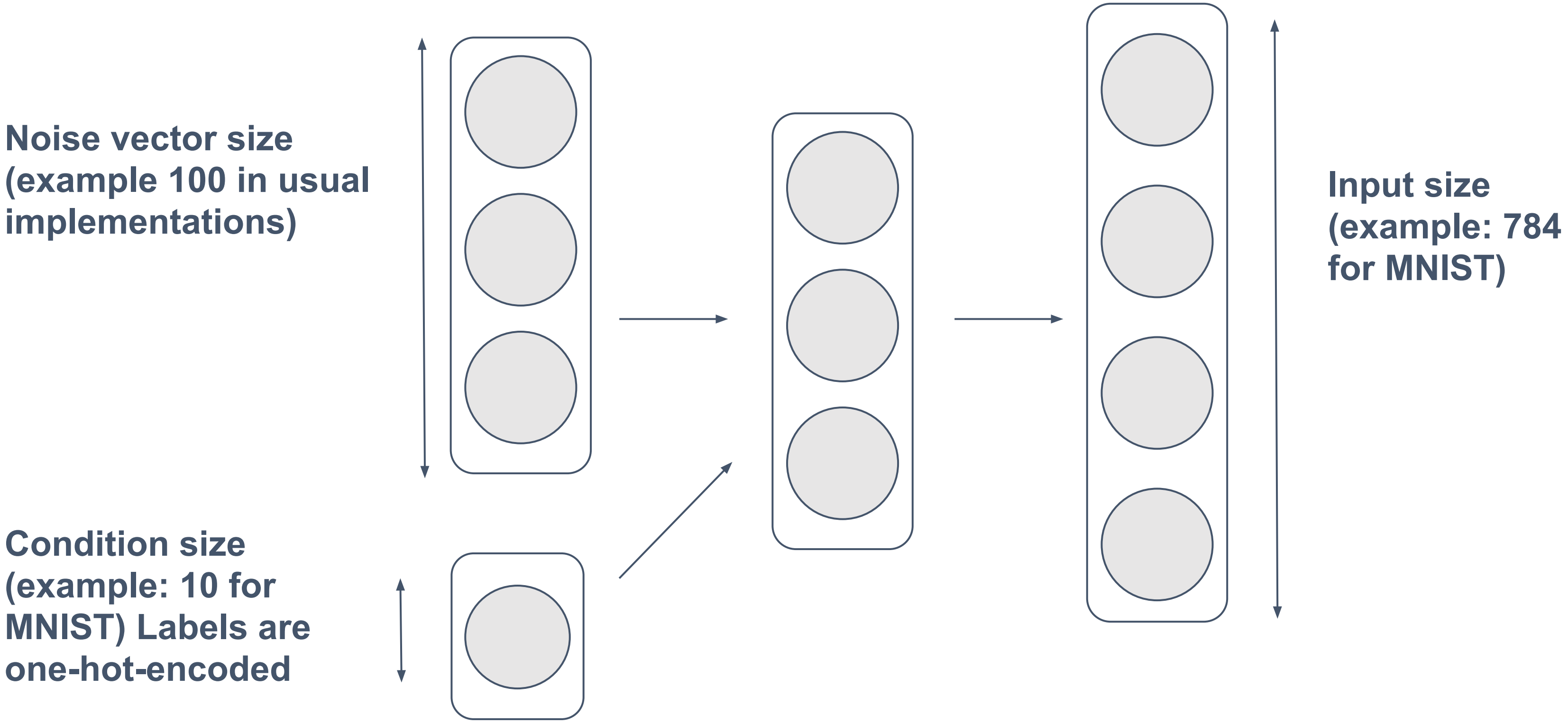
The discriminator (dense NN)



Takes a flattened image and a condition and returns a scalar value.

CGAN architecture

The generator (dense NN)



Takes a vector of random number and a condition and returns a flattened image.

Adapt GAN losses to train the CGAN

- **Discriminator loss:**

$$L_D = -\sum_i^n \log\left(1 - D(G(z_i|c_i)|c_i)\right) + \log\left(D(x_i|c_i)\right)$$

- **Generator loss:**

$$L_G = \sum_i^n \log\left(1 - D(G(z_i|c_i))\right)$$

Warning: To train the discriminator **the same label** must be fed into the generator and the discriminator.

CGAN Practical algorithm

- **Initialize both networks**

- **Training:**
  - Sample a mini-batch of noise vectors, sample a mini-batch of input data and a batch of labels (c_i).
  - Update the discriminator weights with those batches (gradient descent on its loss).
  - Sample a mini-batch of noise vectors and a batch of labels.
  - Update the generator weights with this batch (gradient descent as well).

- **Repeat training phase until convergence**

State of the Art: deepMind 2018

- **Of course, CGAN and DCGAN may be mixed !**

Figure 1: Class-conditional samples generated by our model.

- **This year deepMind** (A. Brock et al., 2018) **proposed an algorithm BigGan that learns the conditional probability of the imageNet dataset (1000 classes !)**

State of the Art: 3D

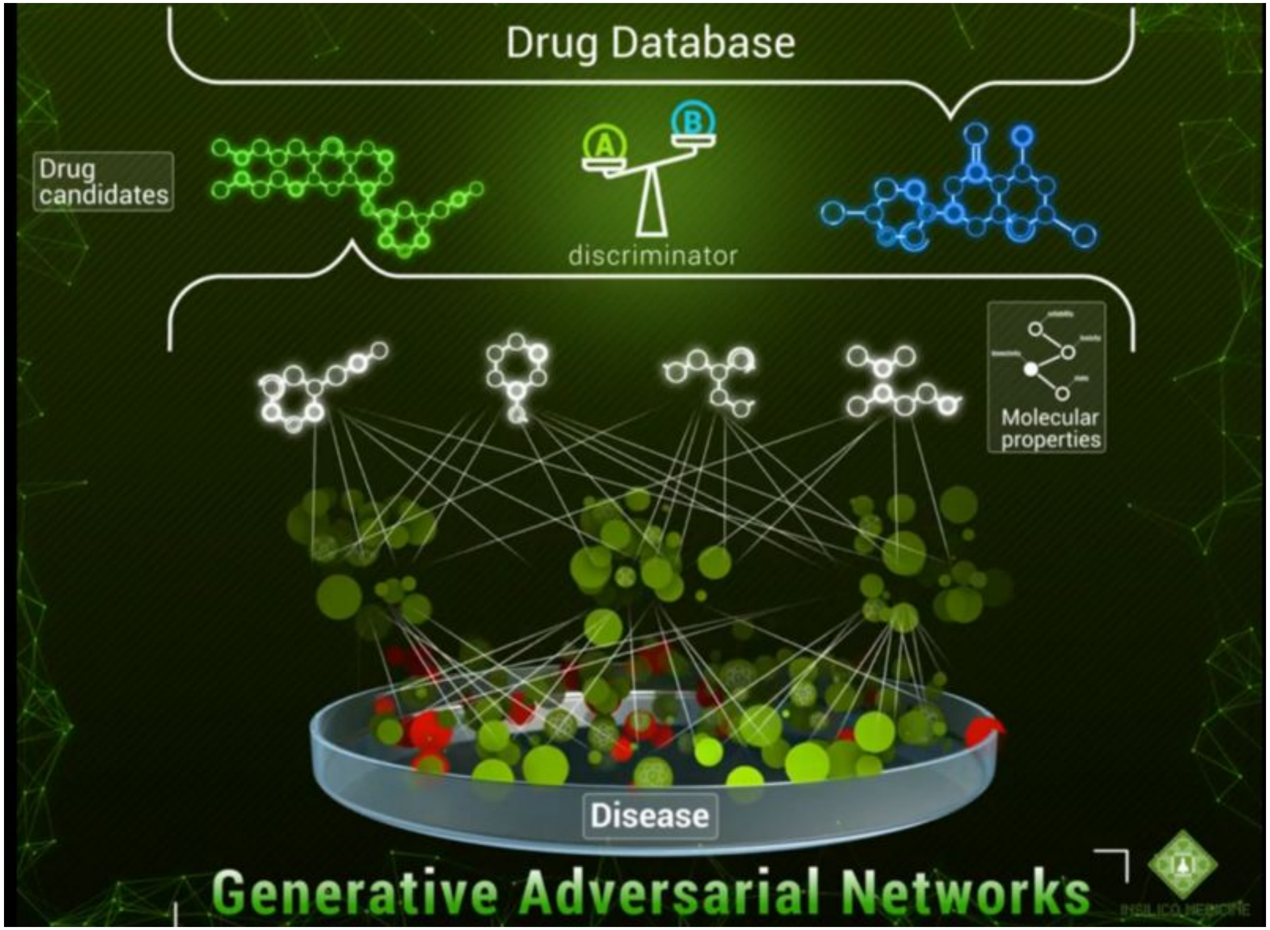
- GAN models can generate 3D objects, 3D environments and even VR environments.
- Wu *et al.* in a ground-breaking article recently published (Jan 2017) showed that it's possible to generate 3D furniture out of simple photos!

- Even more incredible you can even do arithmetic on these 3D models to create **new, never seen before furniture**!

- More state of the art examples: protein research, astronomy

Other applications: drug discovery

See: https://medium.com/neuromation-io-blog/creating-molecules-from-scratch-i-drug-discovery-with-generative-adversarial-networks-9d42cc496fc6



Just a few words about DL libraries

- **Nice blog article about DL libraries :**
  http://www.goldsborough.me/ml/ai/python/2018/02/04/20-17-20-a_promenade_of_pytorch/

- **Most used libraries:**

Tf vs PyTorch

- **Tensorflow: static graphs**
  - Better performance (especially in production)
  - Harder to learn
  - The code is less clear

```
In [1]: import tensorflow as tf
In [2]: x = tf.constant(4)
In [3]: y = tf.constant(2)
In [4]: x + y
Out[4]: <tf.Tensor 'add:0' shape=() dtype=int32>
```

- **PyTorch: dynamic graphs**
  - Easy to learn and use
  - Almost the same synthaxe than numpy
  - Perform auto-differentiation as well
  - Become more popular than tensorflow (especially in the research community)

```
In [1]: import torch
In [2]: x = torch.ones(1) * 4
In [3]: y = torch.ones(1) * 2
In [4]: x + y
Out[4]:
 6
[torch.FloatTensor of size 1]
```

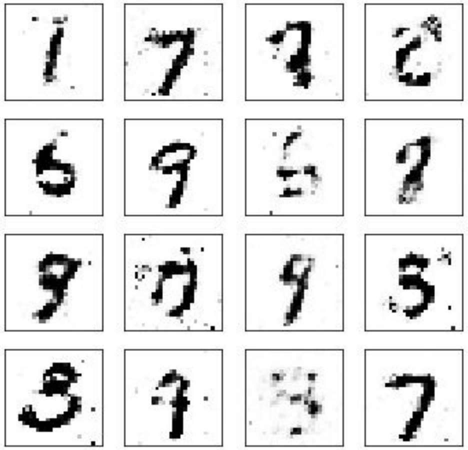
PyTorch advantages

- **Pytorch has an exponentially growing community.**
- **Plenty of high-quality codes in pytorch on GitHub (especially for the GANs).**
- **In opposition to tf, PyTorch tutorials and documentation are very good!**
- **Have a look at:**
  https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html#sphx-glr-beginner-blitz-tensor-tutorial-py

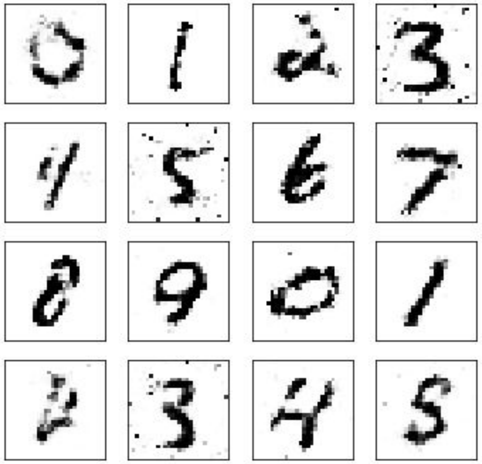
Tutorials

- First part: download, read and complete the notebook "GAN-fill the gaps" that is available on GitHub.

- Second part: take this notebook and adapt it to train a CGAN on MNIST.



Examples of pictures generated by the GAN (we don't control the digits generated).



Examples of pictures generated by the CGAN (we control the digits generated).